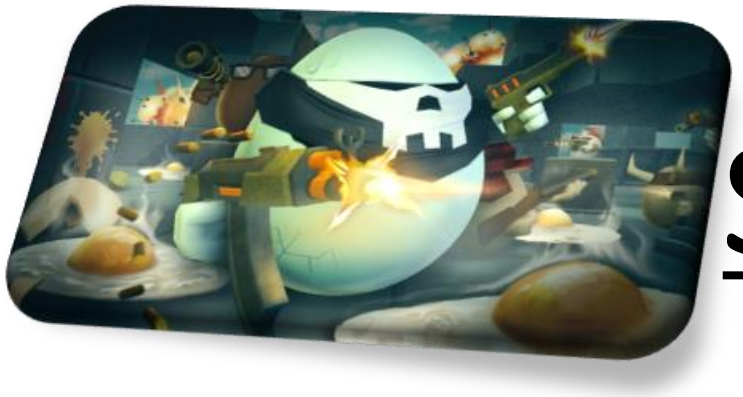
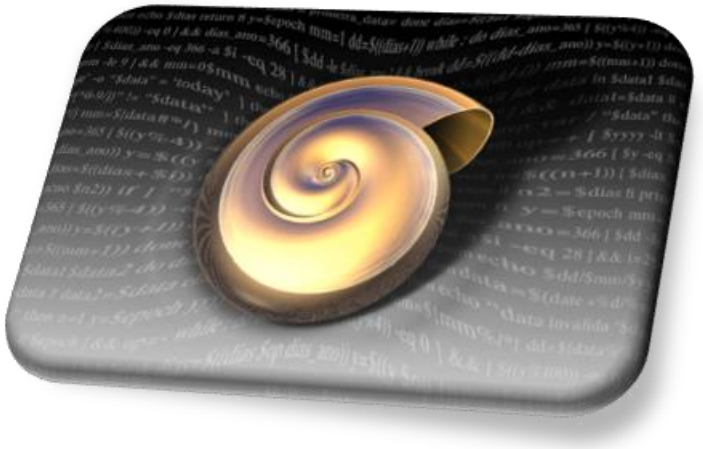
WINDOWS RED TEAM OPERATOR

Malware Analysis & Development w/t Practical & Hands-On



SHELL CODES



WHAT IS SHELL CODE:

In cyber security, a shellcode is "***a small piece of code used as the payload in the exploitation of a software vulnerability"*** to take control of or exploit a compromised machine***.***

It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called ***shellcode***.

Shellcode is *"a set of instructions that executes a command in software to take control of or exploit a compromised machine".*

The word shellcode literally refers to code that starts a command shell -- an instance of a command-line interpreter, such as the shell /bin/sh on Linux or cmd.exe on Windows.

The term now also embraces any bytecode that can be executed once the code is injected into a running application, even if it doesn't spawn a shell.

****Note:*** The function of a payload is not limited to merely spawning a shell.

SHELL CODE Objectives:

Common shellcode objectives includes:

- Targeting program to function in a manner other than what was intended
- Installing a rootkit or Trojan horse,
- Stopping antimalware programs,
- Obtaining sensitive data or downloading files to further compromise the targeted device.

How Does a Shellcode Exploit Work?

Shellcodes are injected into computer memory. After the exploit code causes what would normally be a critical error in the targeted program, the program jumps to the shellcode and is tricked into executing the attacker's commands -- all with the privileges of the process being exploited.

A shellcode exploit consists of two major components:

1. The ***exploitation technique's*** objective is to insert the shellcode and divert the execution path of the vulnerable program to the shellcode so that it can run the code in the payload.
2. The ***payload*** is the component that executes the attacker's malicious code.

For example, shellcode execution can be triggered by overwriting a stack return address with the address of the injected shellcode.

As a result, instead of the subroutine returning to the caller, it returns to the shellcode and spawns a shell.

How Does a Shellcode Exploit Work? (Cont.)

Writing a shellcode to use as a payload to execute something is the easier part of crafting a successful exploit.

Attackers also need to find the address where the shellcode has been stored and gain control of the Extended Instruction Pointer register, which points to the next command, in order to run their exploits.

Simple Shell Code that opens the calculator (i.e. Calc.exe)

```
#include "stdio.h"
unsigned char shellcode[] = "\xeB\x02\xBA\xC7\x93"
                            "\xBF\x77\xFF\xD2\xCC"
                            "\xE8\xF3\xFF\xFF\xFF"
                            "\x63\x61\x6C\x63";
int main ()
{
int *ret;
ret=(int *)&ret+2;
printf("Shellcode Length is : %d\n",strlen(shellcode));
(*ret)=(int)shellcode;
return 0;
}
```

Resource: http://shell-storm.org/shellcode/files/shellcode-567.php

Types of Shellcode Exploits:

Shellcode can be either local or remote:

- **Local shellcode** is used when an attacker has physical access to a machine.
- **Remote shellcode** is used to target a vulnerable process running on another machine to gain access to it across a network.

Single buffers often have limited space where attackers can inject their entire payload into a remote target process. To overcome this restriction, hackers use various techniques, including the following:

- **Staged shellcode.** Shellcode can be executed in stages. The role of the stage one shellcode is to download a more complex and larger stage two shellcode into the process memory and execute it.
- **Egg hunter shellcode.** Similar to a staged shellcode attack, a small "egg hunter" shellcode is injected into the process at a predictable location, which, when executed, searches through the process address space to locate the larger "egg" shellcode, which has been injected at an indeterminate location, and execute it. The egg can have a byte header, which is used as a marker value to help the egg hunter locate and identify it.

Types of Shellcode Exploits:

- **Omelette egg hunter shellcode.**

Like egg hunter shellcode, a small egg hunter shellcode is injected into the process, but it looks for multiple "eggs" (small blocks of code) and rebuilds them into one single block of code (the omelette) that is then executed.

- **Download and execute.**

This type of shellcode instructs the device to download the attacker's malicious file from the internet and execute it. Not only does this enable the shellcode to be small, but it does not need to spawn a new process on the target system.

Programming Errors for Shellcode Exploits:

The most common programming errors used to insert shellcode exploits are *buffer overflows*.

Buffers are temporary areas for data storage.

A buffer overflow occurs when more data is put into a buffer than it has capacity for.

There are two main types of buffer overflows:

1. A stack-based buffer overflow attack occurs when an application's stack -- the area that stores requests -- is exploited.
2. A heap-based buffer overflow attack occurs when an application's memory space is exploited.

Simple Buffer Flow Example:

```c
#include <stdio.h>

int main()
{
    char input[16];
    printf("Enter your password: ");

    // if the password is longer than 16 chars we'll have a buffer overflow;
    scanf("%s", input);
    printf("Your password is %s", input);

    return(0);
}
```

The program reserves 16 bytes of memory for the input, but the input size is never checked. If the user enters a string longer than 16 bytes, the data will overwrite adjacent memory – a buffer overflow.

****Note:** Hackers not always use Buffer Overflow Vulnerabilities as a means to take control of execution, Sometimes its being used to simply bring the application to a crashing halt, which represent a win of sorts for attackers whose objective is some kind of denial of service attack.

Shellcode Usages in Various Vulnerability Exploits:

Apart from Buffer Overflow, Shellcode may be used in other vulnerability exploits, including the following:

- Integer Overflow
- Format String
- Race Condition
- Memory Corruption

Challenges Involved in Writing Shellcode Exploits:

- Anyone writing shellcode needs to have an in-depth understanding of assembly or machine code, C and C++ languages, processor architecture and the targeted OS.

Unlike Linux, Windows does not have a direct kernel interface. The addresses of the functions found in Windows' dynamic link libraries (DLLs) vary from version to version, while Linux has a fixed numbering system for all kernel-level actions.

- Well aware with techniques of using self-decrypting, polymorphic and various static but nonstandard encodings, so that intrusion detection systems cannot detect their shellcode using simple signature matching.

Challenges Involved in Writing Shellcode Exploits:

- The main reason shellcode exploits are possible is because the application or library doesn't correctly validate the data it is handling. OSes allocate specific and finite amounts of memory to hold data, such as variables, values and arrays. These storage areas, called buffers, are generally created at the time a program is loaded or dynamically during program execution. When data exceeding the buffer's capacity is input, it overflows the buffer, and the excess data spills into other memory areas or buffers, overwriting some or all of the contents held in that memory space.
- Software developers need to properly inspect how much data is written into a specific part of a program's code.

Shellcode Techniques for Bypassing Restrictions:

Because most processes filter or restrict the data that can be injected, shellcode often needs to be written to allow/bypass for these restrictions. This includes making the code small, null-free or alphanumeric.

Various ***solutions*** have been found to get around such restrictions, including:

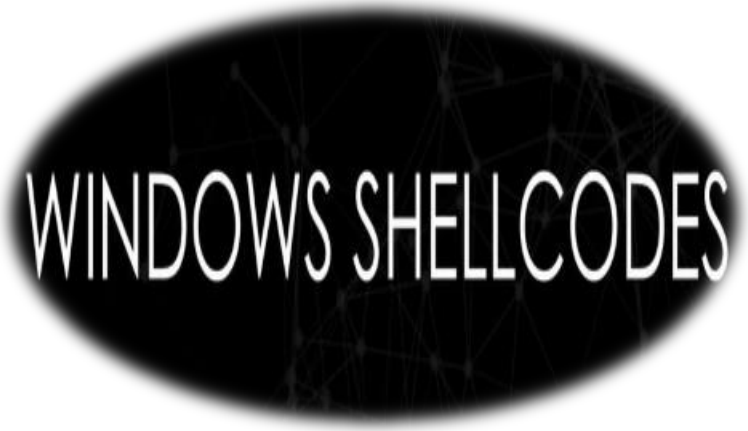
- Design and implementation optimizations to decrease the size of shellcode.
- Implementation modifications to get around limitations in the range of bytes used in the shellcode.
- Self-modifying code that modifies a number of the bytes of its own code before executing them to re-create bytes that are normally impossible to inject into the process.
- Since intrusion detection can detect signatures of simple shellcodes being sent over the network, it is often encoded, made self-decrypting or polymorphic to avoid detection.

Shellcode Exploits Examples:

- A classic attack using shellcode is the exploitation of the ***JpegOfDeath*** vulnerability in gdiplus.dll, which intentionally causes a buffer overflow condition. Anyone who opens a JPEG image created with JpegOfDeath exploit code invokes a buffer overflow, which takes advantage of this condition to inject shellcode into memory that is executed when the overflow occurs.
- A ***watering hole attack*** on a North Korea-related news site used JavaScript to exploit a chain of Google Chrome and Microsoft Windows vulnerabilities (CVE-2019-13720). It injected shellcode, along with a Portable Executable -- Windows executable and object files -- for the shellcode to execute, into the memory of an audio buffer of a WebAudio component.

Shellcode Exploits Examples:

- The 2021 ***Microsoft Office MSHTML Remote Code Execution*** Vulnerability (CVE-2021-40444) enabled remotely hosted shellcode to be loaded into the Microsoft Address Import Tool and be executed without user interaction.



- The ***Purple Fox exploit kit*** exploited a memory corruption vulnerability in Internet Explorer (CVE-2021-26411). It injected shellcode that runs a PowerShell statement to download images from a remote server and execute further PowerShell scripts extracted from the images.

Shell Code Creation & Execution Code Injection